ALFAMA CURSOS

alfamacursos.com.br

Alfama Cursos

Antônio Garcez
Fábio Garcez
Diretores Geral

Antônio Álvaro de Carvalho
Diretor Acadêmico

MATERIAL DIDÁTICO

Produção Técnica e Acadêmica

Marcela Menezes Flores
Coordenadora Geral

Patrícia Queiroz de Meneses
Coordenadora Pedagógica

José Alves Correia NETo
Renata Jacomo Viana
Autoria

Gabriella Caroline Teles Silva
Sabina Regina Conceição Santos
Revisão Textual

Rafael Rezende de Farias
Editoração

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/98.
É proibida a reprodução total ou parcial, por quaisquer meios, sem autorização prévia, por escrito, da ALFAMA CURSOS.

alfamacursos.com.br

Introdução à Linguagem de Programação

Apresentação do Curso

Olá caro aluno!

Seja bem-vindo ao curso de Introdução à Linguagem de Programação, sabendo que é uma matéria de extrema importância para o seu conhecimento. Esta apostila vem complementar as videoaulas.

Lembre-se que apenas pela prática será possível ser um bom programador.

Apresentação do Professor

FÁBIO GOMES ROCHA é formado em Sistemas da Informação e pós-graduado em Engenharia de Sistemas. Atualmente é membro da ABNT (Associação Brasileira de Normas Técnicas), fazendo parte do Comitê 21 - Engenharia de Software e do Comitê 27 - Segurança da Informação. Leciona Técnicas de Desenvolvimento de Sistemas e Linguagem de Programação e na Educação a Distância já trabalhou como desenvolvedor, professor autor e tutor em cursos universitários e técnicos, além de possuir publicações nesta área desde 2006. Além disso, atua na área de Tecnologia da Informação e Comunicação com Desenvolvimento de Sistemas há 17 anos.

Em 2004 publicou um livro sobre Tecnologia para Gestão, intitulado **"Secretariado: do Escriba ao Web Writer".**

Componente Curricular

EMENTA:

Linguagem de Programação; Visual Basic e o .NET; Programando em Visual Basic; Modularizando Códigos VB; Estrutura de Decisão; Estrutura de Repetição; Exceção; Orientação a Objetos; Orientação a Objeto-Herança; Banco de Dados; Componentes para Banco de Dados; GUI.

PÚBLICO-ALVO:

Alunos oriundos do Ensino Médio e universitários que queiram complementar sua formação profissional.

OBJETIVOS GERAIS:

- Conceituar Linguagem de Programação.
- Identificar aplicativos no Visual Basic.
- Definir Propriedades.

HABILIDADES E ATITUDES ESPERADAS:

- Criar aplicativos no Visual Basic ou Pascal.
- Criar Códigos; Procederes, Variáveis e Operações.
- Elaborar Projeto.
- Depurar o Código de um Programa.
- Desenvolver soluções alternativas em Visual Basic como opção para o programador.

Índice

Capítulo 1 - Linguagem de Programação 8
1.2 - Interpretador, Compilador e Bytecode 8
1.3 - Mercado de Software 9
Capítulo 2 - Visual Basic e o .NET 11
2.1 - Plataforma .NET 11
2.2 - Arquitetura do .NET 11
2.3 - Visual Studio e Visual Basic 12
2.3.1 - Ferramenta Visual Studio 12
Capítulo 3 - Programando em Visual Basic 14
3.1 - Palavras Reservadas 14
3.2 - Comentários 14
3.3 - Tipos de Valores 15
3.4 - Operadores 16
3.5 - Criando o Primeiro Programa em Visual Basic 18
Capítulo 4 - Modularizando Códigos no Visual Basic 20
4.1 - Sub-rotina 20
4.2 - Função 20
Capítulo 5 - Estrutura de Decisão 23
5.1 - Decisão Estendida 23
Capítulo 6 - Estrutura de Repetição 27
Capítulo 7 - Exceção 30
7.1 - Tipos de Exceção 30
7.2 - Cath 31
Capítulo 8 - Orientação a Objetos 33
8.1 - Programação Orientada a Objetos 33
8.1.1 - Criando Classes no Visual Basic 33
8.2 - Membros de uma classe 33
Capítulo 9 - Orientação a Objetos-herança 36
9.1 - Herança na Prática 36
Capítulo 10 - Banco de Dados 39
Capítulo 11 - Componentes para Banco de Dados 41
11.1 - Biblioteca ADO.NET 41
11.1.1 - Vantagens 41
11.2 - Componentes ADO.NET 41
11.2.1 - Componentes do Tipo SQL 42
11.2.2 - Componentes do Tipo OLEDB 42
11.2.3 - Componentes para Manipulação de Dados 42
Capítulo 12 - GUI 44
Referências 47

1 - LINGUAGEM DE PROGRAMAÇÃO

O que é uma Linguagem de Programação?

Uma Linguagem de Programação converte algoritmo em uma determinada linguagem em instruções que a máquina possa entender. Ou seja, converte uma codificação em linguagem de máquina.

Historicamente temos:

- 1º Compilador: 1952 criado por Grace Hopper.
- A linguagem A-0.
- Fortram – 1954.
- B-0 – 1957 --> Que gerou o Flow-Matic – que em 1958 deu origem ao Cobol, utilizado até os dias atuais em instituições financeiras.
- Lisp – 1958.
- Em 1964 surgiu o Basic (1975 --> Basic Microsoft --> QBasic --> Visual Basic).

Uma pesquisa efetuada na InterNET em 2010 apresentou o seguinte resultado das linguagens mais utilizadas:

Figura 1: Preferência em Linguagens de Programação.

1.2 - INTERPRETADOR, COMPILADOR E BYTECODE

Existem atualmente diversas formas de gerar programas, mas três formas se destacam, sendo elas:

- **Interpretador:** O interpretador é um programa que executa o código criado em determinada linguagem. Esta execução converte em tempo real o código em linguagem de máquina, o que torna o processo um pouco lento, pois toda vez que o código for ser executado, este deve gerar o código de máquina.
- **Compilador:** O compilador serve para converter a codificação em linguagem de máquina, desta forma o programa fica muito mais rápido, pois já está em um formato que a máquina entende. Este programa, normalmente, não é multiplataforma, ou seja, se criarmos uma versão compilada para o Windows, provavelmente esta não funcionará no Linux, portanto devemos ter uma versão para cada sistema operacional.
- **Bytecode:** Este formato é utilizado por linguagens como o Java e o .NET, que

para funcionar deve ter a máquina virtual ou o framework, tornando o sistema multiplataforma, pois além de rodar em diversos sistemas operacionais (Basta ter a máquina virtual instalada) pode rodar em diversos dispositivos como celular, tablet e etc.

O funcionamento de qualquer linguagem compilada ou bytecode é apresentado na figura 2.

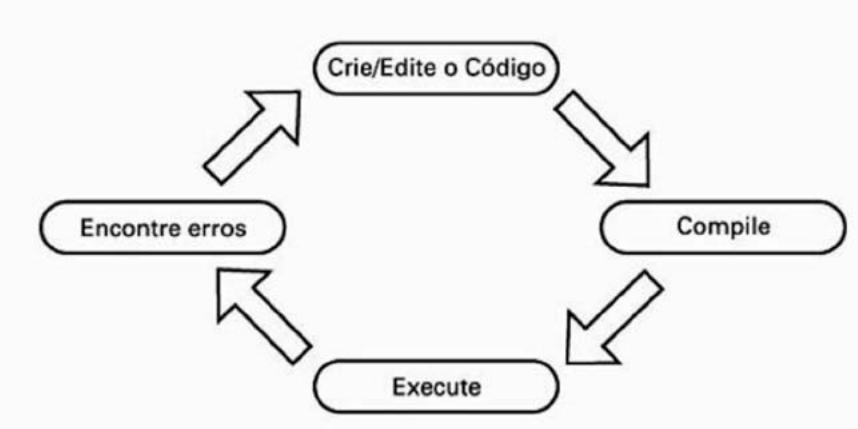


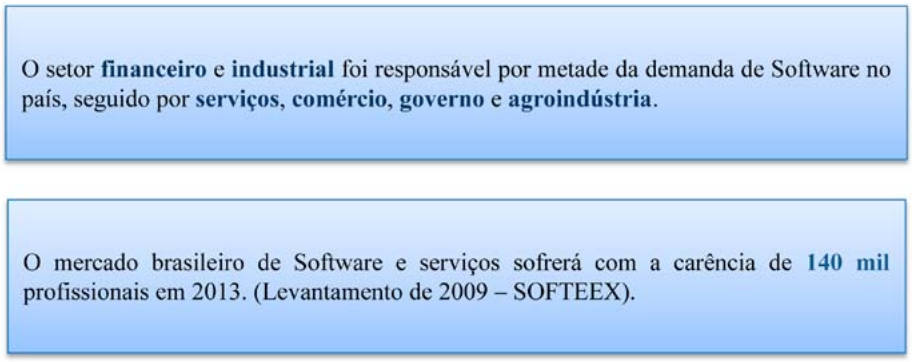
Figura 2: Processo de criação de um programa.

1.3 - MERCADO DE SOFTWARE

Segundo pesquisas, o mercado de software gira em torno de bilhões no mundo. Segundo pesquisa temos:

1º Lugar no mercado de Software.
Faturamento: US$ 349,7 bilhões, em 2009.
2º Lugar no mercado de Software.
Faturamento: US$ 71,7 bilhões.
12º Lugar no mercado de Software.
Faturamento: US$ 15,38 bilhões, 2009.

O Brasil ainda é o 12º devido à falta de incentivo fiscal e as dificuldades do mercado nacional, mesmo assim, é um mercado que fatura 15,38 bilhões somente em 2009. Dados podem ser encontrados no site da SOFTEEX.
Além disso, temos os seguintes dados referentes aos setores que possuem mais demanda de software no Brasil.


O setor financeiro e industrial foi responsável por metade da demanda de Software no país, seguido por serviços, comércio, governo e agroindústria.
O mercado brasileiro de Software e serviços sofrerá com a carência de 140 mil profissionais em 2013. (Levantamento de 2009 – SOFTEEX).

Recordando

Neste capítulo, você aprendeu o que é uma Linguagem de Programação, para que serve e viu o que o mercado de Software além de promissor, é um dos maiores do mundo. Nos próximos capítulos, você aprenderá sobre o Visual Basic e seu funcionamento.

2 - VISUAL BASIC E O .NET

2.1 - PLATAFORMA .NET

O .NET Framework é uma iniciativa da empresa Microsoft, que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações.

Todo e qualquer código gerado para o .NET, pode ser executado em qualquer dispositivo que possua um Framework de tal plataforma.

A Plataforma .NET foi concebida sob uma plataforma robusta e em camadas. O . é uma camada de software que fica entre o Sistema Operacional e as aplicações, conforme pode ser visto na figura 3.

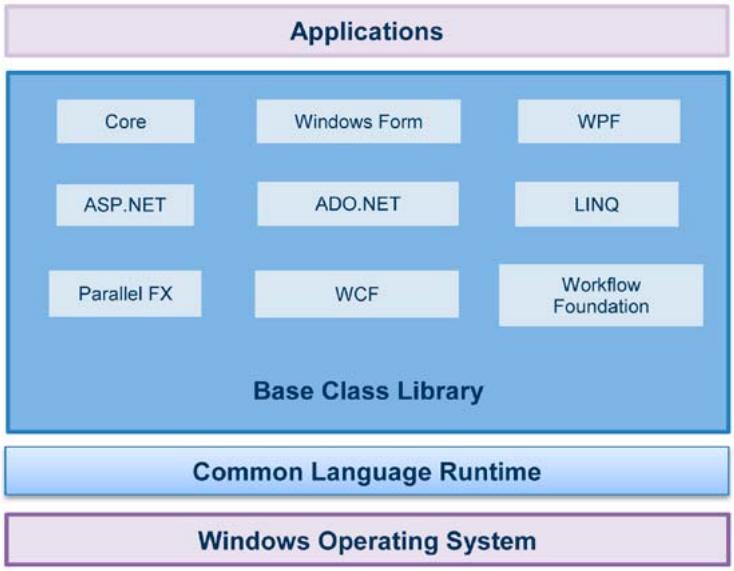


Figura 3: Arquitetura .NET.

2.2 - ARQUITETURA DO .NET

a) Camada Inferior: Common Language Runtime (CLR)

Executa a maior parte das tarefas e faz a comunicação com o Sistema Operacional.

b) Camada Superior: Base Class Library (BCL)

Contém os tipos, as interfaces e as bibliotecas de classes utilizadas para a criação de uma aplicação. Dispensando o esforço e o consumo de tempo para que estas sejam codificadas pelo programador da aplicação.

O .NET é standalone, ou seja, todo CLR e toda BCL estão contidos na versão da plataforma e, desta forma, não necessita que as versões anteriores estejam instaladas.

2.3 - VISUAL STUDIO E VISUAL BASIC

O Visual Studio é o pacote da Microsoft que possui versões diferenciadas. As versões Express são gratuitas, podendo ser feito download gratuito diretamente do site da Microsoft.

Visual Studio 2010 Express para Windows Phone

Oferece tudo o que você precisa para começar a criação de aplicativos Windows Phone.

Visual Web Developer 2010 Express

É um editor de desenvolvimento web completo para trabalhar com padrões web, JavaScript e ASP.NET

Visual Basic 2010 Express

Visual Web Developer 2010 Express

Visual C++ 2010 Express

2.3.1 - FERRAMENTA VISUAL STUDIO

A família do Microsoft Visual Studio é composta por:

- Visual Studio 2010 Premium.
- Visual Studio 2010 Professional.
- Visual Studio 2010 Ultimate.
- Visual Teste Professional 2010.

Visual Studio Team Foundation Server
Visual Studio Test Professional 2010
Visual Studio 2010 Ultimate
Visual Studio 2010 Professional

Figura 4: Família de produtos do Visual Studio.

Recordando

Neste capítulo, você aprendeu o que é a Plataforma .NET, viu que ela serve como meio de padronizar o formato de desenvolvimento entre as diversas linguagens da Microsoft, como é possível utilizar gratuitamente o Visual Basic e como fazer o download.

3 - PROGRAMANDO EM VISUAL BASIC

O Visual Basic foi criado com base no Microsoft Basic, que é uma das primeiras ferramentas que a Microsoft lançou no mercado, antes de se aventurar no mercado de Sistemas Operacionais, sendo uma linguagem padrão em diversos equipamentos da década de 80. Com a chegada da Interface Gráfica, o Basic evoluiu para Visual Basic e tornou-se uma das ferramentas mais populares para desenvolvimento visual.

No capítulo 2, vimos como fazer o download, agora vamos iniciar a utilização da ferramenta. A variável é uma das principais formas de armazenar dados de forma temporária em uma linguagem de programação. No Visual Basic não é diferente, só temos regras para fazer uso das variáveis.

Para declarar uma variável no Visual Basic, utilizamos a palavra-chave "Dim", como pode ser visto a seguir:

Declaração de variáveis:
Dim nome As tipo de dados
Exemplos:
Dim idade As Byte
Dim endereco As String
Dim num1, num2, result As Integer
Dim numero As Ushort, Complemento As String

Além das variáveis, temos as constantes, cuja principal diferença é que esta não pode ser alterada durante o uso do sistema. É como se fosse uma memória RAM, sendo um espaço alocado na memória para armazenamento de um valor, que após a sua atribuição não pode ser alterado.

Uma constante tem:

- Nome identificador.
- Endereço inicial de alocação na memória.
- Tamanho do espaço de alocação na memória.
- Tipo de Dados.

Declaração de constantes:

Const nome As tipo de dados = valor
CUIDADO COM PALAVRAS RESERVADAS!

3.1 - PALAVRAS RESERVADAS

As palavras reservadas não podem ser usadas como identificadores de variáveis, constantes, procedimentos, módulos, sub-rotinas, funções, objetos, entre outros.

Exemplos: Dim, next, if, namespace, …

Para uma lista completa visite: http://msdn.microsoft.com/en-us/library/dd409611.aspx

3.2 - COMENTÁRIOS

São utilizados para descrever o código com objetivo de documentar e facilitar o entendimento do programa.

Os comentários são prefixados com apóstrofe.

O compilador VB ignora todos os caracteres após a apóstrofe até o final da linha de código.

Exemplo:

Dim x As Integer ‘Declaração de Variáveis

Todo comentário fica em verde

3.3 - TIPOS DE VALORES

As variáveis podem assumir valores, podendo ser numéricos e alfanuméricos. Entre os tipos temos:

Boolean – Depende de plataforma --> Faixa (True/False) Verdadeiro/Falso

- Caracteres

Os tipos de caracteres servem para manipular dados alfanuméricos, ou seja, contendo letras e números que não serão utilizados para efetuar cálculos.

- Char (caracter simples)

Define variáveis ou constantes que armazenam o código em 2 bytes de um caracter na codificação Unicode.

Valor Default: Zero

- String (cadeia de caracteres)

Define um ponteiro para uma cadeia de caracteres codificados em Unicode.

A cadeia de caracteres pode ter até aproximadamente 231 caracteres.

Valor Default: Nothing (sem endereço).

O programador deve escolher o tipo que ocupe a menor quantidade de memória em bytes e seja capaz de armazenar a faixa de valores possíveis de um determinado dado a ser armazenado em uma variável ou constante.

Tipo	Tamanho
Byte	1 byte
SByte	1 byte
Short	2 bytes
UShort	2 bytes
Integer	4 bytes
UInteger	4 bytes
Long	8 bytes
ULong	8 bytes
Single	4 bytes
Double	8 bytes
Decimal	10 bytes

Tipo	Descrição	Tamanho	Faixa de Valores
Date	Define variáveis ou constantes que armazenam data (MM/DD/AAAA), Hora ou data e hora. Delimitador: # (#10/20/2010)	8 Bytes	00:00:00 (meia noite) de 01/01/0001 a 11:59:59 (PM) de 31/12/9999.
Object	Define um ponteiro que armazena o endereço de refrência a qualquer valor ou objeto	4 bytes (Plataforma de 32 bits) 8 bytes (Plataforma de 64 bits)	Valores do tipo do objeto
Structure	Permite que o programador defina estruturas com membros de diferentes tipos	Depende da plataforma	Cada membro da estrutura tem valores definidos pelo respectivo tipo e independente dos demais membros.

3.4 - OPERADORES

Além dos tipos, um programador deve conhecer os operadores, que servem para manipular dados, podendo ser:

- **Operador de Atribuição**

- É feita através do sinal de =

Exemplos:

```
idade = 10
disciplina = "Desenvolvimento de Software"
```

– Declaração de Variáveis e atribuição de Valor.

```
Dim  dependentes As Byte = 1
```

– Declaração de Constantes e atribuição de Valor.

```
Const  pi As Double = 3.1415
```

Operador	Exemplo
=	x = y
+=	x += y equivale x = x + y
-=	x -= y equivale x = x – y
*=	x *= y equivale x = x * y
/=	x /= y equivale x = x / y
\=	x \= y equivale x = x \ y
^=	x ^= y equivale x = x ^ y
<<=	x <<= y equivale x = x << y
>>=	x >>= y equivale x = x >> y
&=	x &= y equivale x = x & y

- **Operador Aritmético**

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
\	Parte inteira da Divisão
mod	Resto da Divisão
^	Potência

- **Operador de Comparação**

Operador	Descrição
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
<>	Diferente
=	Igual
Is	Retorna **True** caso dois nomes de objetos (ponteiros) apontam para o mesmo objeto. Retorna **False** caso contrário.
IsNot	Retorna **True** caso dois nomes de objetos (ponteiros) não apontam para o mesmo objeto. Retorna **False** caso contrário.
Like	Retorna **True** caso um padrão de string esteja contido em uma string. Retorna **False** caso contrário.

- **Operadores Lógicos**

Operador	Descrição
And	Operação **E**
Not	Operação **Not**
OR	Operação **Ou**
Xor	Operação **Ou exclusivo**
AndAlso	Operação **E** -- Se a primeira expressão em um AndAlso expressão é avaliada como False, em seguida, a segunda expressão não será avaliada porque ele não é possível alterar o resultado final, e AndAlso Retorna False.
OrElse	Operação **Ou** -- Se a primeira expressão em um OrElse expressão é avaliada como True, em seguida, a segunda expressão não será avaliada porque ele não é possível alterar o resultado final, e OrElse Retorna True.

- **Operador de Concatenação**

Operador	Descrição
&	Concatena duas Strings
+	Concatena duas Strings (equivale ao operador &)

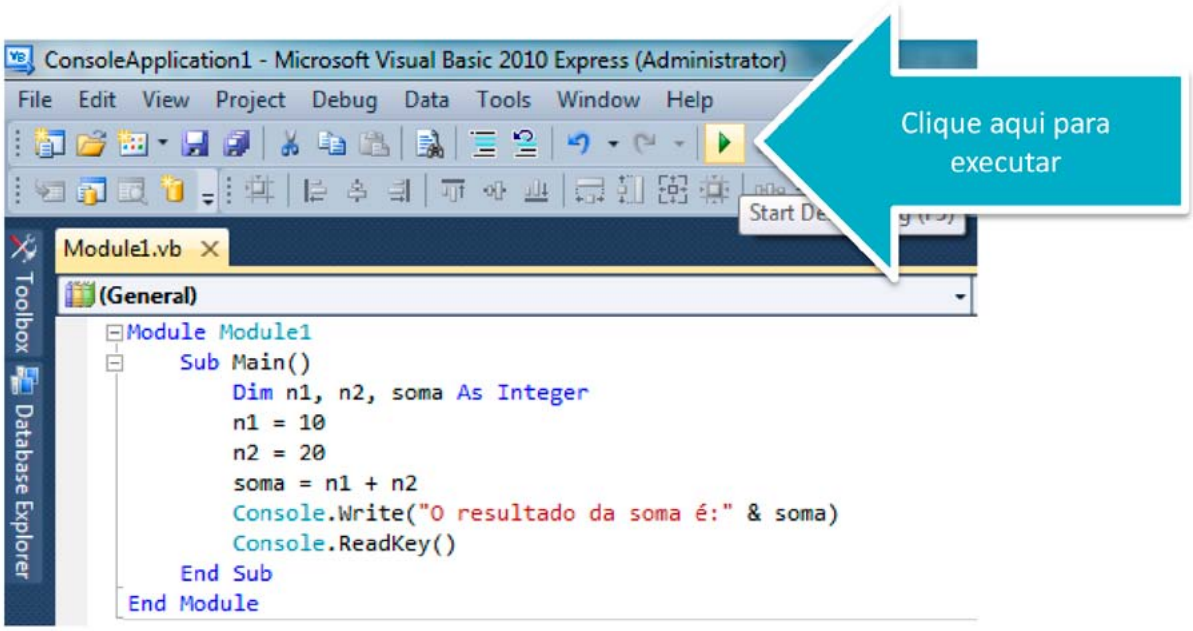
3.5 - CRIANDO O PRIMEIRO PROGRAMA EM VISUAL BASIC

Abra o Visual Basic Expresso --> New Project --> Console Application --> A codificação aberta será:

```
        Module Module1
            Sub Main()
            End Sub
        End Module
```

Mude para:

```
Module Module1
    Sub Main()
     Dim n1, n2, soma As Integer
        n1 = 10
        n2 = 20
        soma = n1 + n2
        Console.Write("O resultado da soma é:" & soma)
                Console.ReadKey()
            End Sub
        End Module
```



O programa cria 3 variáveis do tipo inteiro. O comando Console.Write serve para exibir dados na tela e o Console.ReadKey() fica aguardando o usuário pressionar qualquer tecla para finalizar.

Recordando

Neste capítulo, você aprendeu sobre variáveis, constantes e tipos de valores no Visual Basic e viu ainda como criar seu primeiro programa. Agora coloque em prática o que você aprendeu.

4 - MODULARIZANDO CÓDIGOS NO VISUAL BASIC

A modularização de código dentro de uma linguagem de programação é de extrema importância, visto que sem este recurso, teríamos centenas de linhas seguidas para poder entender um programa. A modularização permite criar partes de código para tarefas, desta forma fica muito mais simples a manipulação e a manutenção da codificação. Outro fator importante é a possibilidade de trabalho em grupo, já que pode ser possível um programador criar um módulo enquanto outro programador cria outro módulo.

4.1 - SUB-ROTINA

É um método que executa um bloco de código, mas não retorna valor, sendo uma forma simples de modularizar seus programas no Visual Basic.

Sintaxe de uma sub-rotina:

```
Sub   nome(ByVal NomeParametro1   As Tipo, ...., ByVal NomeParametro1)

        <bloco de código>
End Sub
```

Vamos criar nosso primeiro programa modularizado. Crie um novo projeto no Visual Basic com a seguinte codificação:

```
Module Module1
   Sub Main()
      Console.Write("CAPÍTULO 4 – Modularizando códigos VB")
      Console.ReadKey()
   End Sub
End Module
```

Execute e veja o resultado. Agora vamos alterar a codificação para um sistema modularizado.

```
Module Module1
   Sub ExibirMSG(ByVal disciplina As String)
      Console.Write("Disciplina: " & disciplina)
   End Sub
   Sub Main()
      ExibirMSG("Introdução a Linguagem de Programação")
      Console.ReadKey()
   End Sub
End Module
```

Ao executar, note que o resultado é semelhante, porém mudamos a forma de trabalhar. Caso queira pode-se utilizar a sub-rotina ExibirMSG para escrever qualquer texto que desejar.

Agora tente transformar seu programa de soma em um programa modularizado, vai notar algumas vantagens, pois poderá ter sub-rotinas para cada tipo de cálculo, como veremos posteriormente.

4.2 - FUNÇÃO

É um método que executa um bloco de código, mas retorna valor. Note que é semelhante à sub-rotina na forma de criar e utilizar, com a principal diferença no retorno de valor, através do return.

- Sintaxe de uma função

```
Function nome(ByVal NomeParametro1   As Tipo, ...., ByVal NomeParametro1)

      <bloco de código>

      Return expressão

  End Function
```

Vamos melhorar nosso programa de sub-rotina, implementando uma função.

```
Module Module1
    Sub ExibirMSG(ByVal disciplina As String)
        Console.WriteLine("Disciplina: " & disciplina)
    End Sub
    Function Quadrado(ByVal valor As Double)
        Return valor * valor
    End Function
    Sub Main()
    ExibirMSG("Introdução a Linguagem de Programação")
        Console.WriteLine("A área do quadrado é:" & Quadrado(3.5))
        Console.ReadKey()
    End Sub
End Module
```

Ao executar, note a utilização de sub-rotina e de funções em seu programa.

Recordando

Neste capítulo, você aprendeu como modularizar seus códigos no Visual Basic, fazendo uso das sub-rotinas e das funções, e viu também exemplos práticos de como fazer tal uso. Agora pratique utilizando o que você já aprendeu.

5 - ESTRUTURA DE DECISÃO

O processo condicional também é um processo de decisão ou escolha de alternativas. Utilizamos diariamente o sistema condicional, como exemplo:

Se chover então vou de carro senão vou de bicicleta.

Para tomarmos decisão, normalmente analisamos alguns pontos para verificar se é verdadeiro ou falso, no caso da chuva eu posso abrir a janela e ver se está chovendo ou como está o tempo. Com isso, estou fazendo um teste de condicional.

Em algoritmo, o teste condicional é feito da seguinte forma:

```
Se  <condição> Então
    <bloco de código_Verdade>
Fim se
```

Dentro do Visual Basic a aplicação é feita da seguinte forma:

```
If  <condição> Then
    <bloco de código_Verdade>
End If
```

Exemplo:

```
Dim n1 As Integer = 20
IF n1 >  10  Then
    Console.WriteLine("Número maior que 10")
End If
```

5.1 - DECISÃO ESTENDIDA

A decisão estendida utiliza o bloco contínuo else para fazer uma nova condição:
Em algoritmo:

```
Se  <condição> Então
       <bloco de código_Verdade>
   Senão
       <bloco de código_Falso>
Fim se
```

Em VB:

```
If  <condição> Then
        <bloco de código_Verdade>
   Else
        <bloco de código_Falso>
End If
```

Faça o exemplo a seguir:

Exemplo: Receber um número e informar se é par ou ímpar.

```
Sub Main()
     Dim n1 As Integer
     Console.WriteLine("Número")
      n1 = Console.ReadLine()
      IF n1 mod 2 =  0  Then
          Console.WriteLine("Número é Par")
      Else
           Console.WriteLine("Número é Ímpar")
```

```
    End If
        Console.ReadKey()
End Sub
```

Outra forma é a extensão da estrutura se, utilizando o se alinhado, ou seja, se junto com se.

```
If  <condição 1> Then
      <bloco de código_1>
  Else
      If  <condição 2> Then
            <bloco de código_2>
          Else
          <bloco de código_Falso>
      End If
End If
```

Exemplo: Receber a idade de uma pessoa e informar se é Criança, Adolescente, Adulto ou Idoso.

```
Sub Main()
   Dim idade As Byte
   Console.WriteLine("Idade: ")
   Idade = Console.ReadLine()
   IF idade < 12 Then
       Console.WriteLine("Criança")
    ElseIF idade < 18 Then
       Console.WriteLine("Adolescente")
    ElseIF idade <= 60 Then
       Console.WriteLine("Adulto")
    Else
       Console.WriteLine("Idoso")
End If
   Console.ReadKey()
End Sub
```

O se não é a única estrutura condicional existente, temos também o caso, que é outra estrutura de decisão mais simples e permite diversas saídas, o chamado múltiplas decisões.

Em algoritmo:

```
escolha <variável>
caso <condição>
 <Tarefa a ser realizada>
caso <condição>
  <Tarefa a ser realizada>
outrocaso
   <Tarefa a ser realizada>
fimescolha
```

Em VB temos:

```
Select Case  <condição 1>
    Case <valor ou intervalo ou lista de valores>
            <bloco de instrução 1>
    Case <valor ou intervalo ou lista de valores>
            <bloco de instrução 2>
    Case <valor ou intervalo ou lista de valores>
            <bloco de instrução 3>
     ......
```

```
        Case Else
            <bloco de Instrução N>
End Select
```

Vamos ver um exemplo completo utilizando múltiplas decisões.

```
Sub Main()
        Dim N1 As Byte
        Console.Write("Digite um número de 0 a 99: ")
        N1 = Console.ReadLine
        Select Case (Numero)
            Case 0 To 9
                Console.WriteLine("O número está no intervalo de 0 a 9 inclusive.")
            Case 10 To 99
                Console.WriteLine("O número está no intervalo de 10 a 99 inclusive.")
            Case Else
                Console.WriteLine("O número não está no intervalo de 0 a 99 inclusive.")
        End Select
        Console.ReadKey()
    End Sub
```

Recordando

Neste capítulo, você aprendeu como tomar decisão dentro do Visual Basic. Então coloque em prática tudo o que você já aprendeu.

6 - ESTRUTURA DE REPETIÇÃO

A estrutura de repetição é utilizada quando desejamos que um conjunto de instruções seja executado um número definido ou indefinido de vezes.

Os laços de repetição são utilizados quando desejamos que um determinado conjunto de instruções seja executado um número de vezes, sendo finito ou infinito.

Uma das principais estruturas de repetição é o for, que em algoritmo é apresentado desta forma:

```
Para <Valor Inicial> Até <Valor Final> <Passo n>  Faça
      <bloco instrução>
Fim Para
```

Exemplo:

```
Para i de 1 Até 10 passo 1 Faça
      escreva(i)
Fim Para
```

Em VB:

```
    For contador = valor inicial To valor final
        Bloco de código
   Next contador
```

Exemplo em Visual Basic:

```
Sub Main()
Dim  i  As Byte
       For i = 1 To 10
          Console.WriteLine(i)
       Next  i
       Console.ReadKey()
End Sub
```

Outro exemplo, desta vez pulando a numeração de 2 em 2.

```
Sub Main()
Dim  i  As Byte
       For i = 1 To 10 Step 2
          Console.WriteLine(i)
       Next  i
       Console.ReadKey()
End Sub
```

Outra estrutura de repetição que veremos é o enquanto (while). Este verifica a condição e enquanto esta não for verdadeira, continua executando o código.

Em algoritmo:

```
Enquanto <condição> Faça
      <bloco instrução>
Fim Enquanto
```

Exemplo:

```
      x <-1
```

```
      y <- 5
Enquanto  x < y Faça
    x <-x + 2
    y <- y + 1
Fim Enquanto
Em VB:
    While (condição)
        Bloco de instrução
   End While
```

Exemplo:

```
      x = 1
      y = 5
      While  (x < y)
    x = x + 2
    y = y + 1
End While
```

Um exemplo completo utilizando VB:

```
Sub Main()
       Dim N1 As Integer
       Dim x As Integer = 0
       Console.Write("Digite um número:  ")
       N1 = Console.ReadLine
      While  (n1 <> 0)
    x = x + 1
            Console.Write("Digite um número:  ")
            N1 = Console.ReadLine
End While
        Console.Write("Quantidade de números Digitados:  " & x)
        Console.ReaKey()
End Sub
```

Recordando

Neste capítulo, você aprendeu como repetir dentro do Visual Basic. Então coloque em prática tudo o que você aprendeu.

7 - EXCEÇÃO

Ao desenvolver sistemas podem ocorrer erros e alguns desses erros são possíveis de controlar e de saber onde vão ocorrer. A exceção ajuda os programadores a criar tratamento de erros, antes de eles ocorrerem.

Exceção é um erro no fluxo normal de execução do código de um programa de uma aplicação.

O .NET Framework oferece uma forma unificada de identificar e tratar exceções entre todas as linguagens de programação .NET (BALENA, 2006).

Há dois tipos de exceções:

- Exceções de Hardware.
- Exceções de Software.

a) Exceção de Hardware

São causadas por erros do programa a nível de Hardware.
Exemplo: Violação de acesso a endereços de memória.

b) Exceção de Software

São causadas por erros do programa a nível de Software.
Exemplo: Atribuir um valor incompatível a uma variável.

Divisão por zero.

O tratamento de exceções estruturadas (Structured Exception Handling - SEH) é uma estratégia para codificar o programa de uma aplicação para tratar tanto as exceções de Hardware quanto as exceções de Software.

Dessa forma, o código para tratamento de exceções se torna unificado e mais consistente por tratar tanto as exceções de Hardware quanto às exceções de Software.

O tratamento de exceções estruturadas é codificado para proteger seções do código que estiverem propensas a gerarem exceções como, por exemplo, cálculos, acesso a banco de dados, operações com arquivos e etc. (MACKENZIE e SHARKEY, 2003).

7.1 - TIPOS DE EXCEÇÃO

Os principais tipos de exceções são:

- ArgumentException
- ArgumentNullException
- ArgumentOutOfRangeException
- IndexOutOfRangeException
- NotImplementedException
- OutOfMemoryException
- OverflowException
- FileNotFoundException

As palavras-chave “Try... End Try” delimitam o bloco de códigos protegidos para tratar as exceções que ocorrerem.

Sintaxe:
```
Try
   ...
   Catch ex As Exception
   ...
End Try
```

7.2 - CATH

Marca a seção do bloco que captura e trata as exceções que podem eventualmente ocorrer. Em geral, usa-se uma seção Catch para capturar cada tipo de exceção e uma seção Catch, chamada genérica, no final para capturar todos os demais tipos de exceções que não foram capturados pelas seções Catch anteriores.

Exemplo: Com Captura e Tratamento de Exceção.

```
Sub Main()
        Dim i As Decimal = 10
        Dim j As Decimal = 0
        Dim Resultado As Decimal
       Try
         Resultado = i / j
         Catch ex As Exception
            ` Captura Genérica
            Console.WriteLine("Houve um erro de exceção de divisão por zero.")
        End Try
        Console.ReadKey()
End Sub
```

Recordando

Neste capítulo, você aprendeu como tratar seus erros utilizando o sistema de exceção. Este é um conceito novo apresentado aqui, então pratique bastante para fixá-lo em sua memória.

8 - ORIENTAÇÃO A OBJETOS

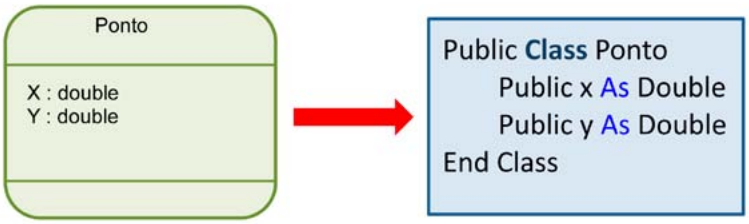
Orientação a objetos, apesar de não ser nada novo, tem ganhado muita força após o período da criação dos padrões de análise Orientada a Objetos com recursos de UML. Aos programas utilizando Orientação a Objetos, nos deparamos com utilização de Classes e Objetos, mas o que são e qual a diferença entre eles?

Definir uma **classe** significa, em programação, definir um modelo com os atributos (dados) e **métodos** (as funções) associados a essa entidade. Já um objeto constitui uma entidade concreta com tempo e espaço de existência, ou seja, um objeto é a representação de uma entidade real de uma determinada classe.

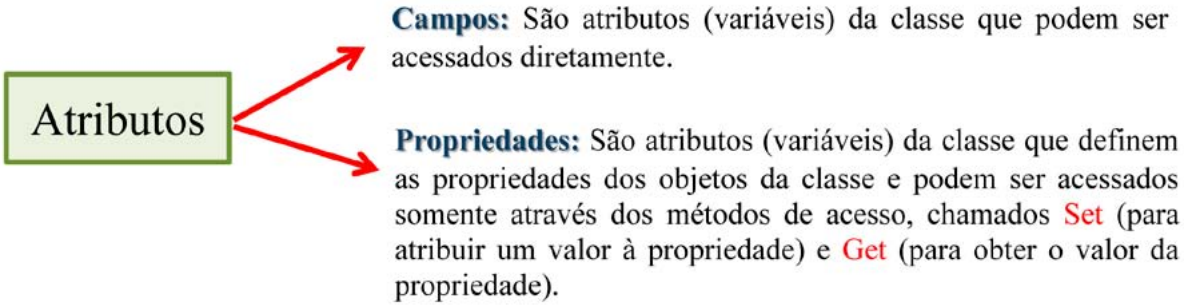
8.1 - PROGRAMAÇÃO ORIENTADA A OBJETOS

8.1.1 - CRIANDO CLASSES NO VISUAL BASIC

```
Public Class NomedaClasse
    ` definição dos membros da classe
End Class
```



8.2 - MEMBROS DE UMA CLASSE


Atributos
Campos: São atributos (variáveis) da classe que podem ser acessados diretamente.
Propriedades: São atributos (variáveis) da classe que definem as propriedades dos objetos da classe e podem ser acessados somente através dos métodos de acesso, chamados Set (para atribuir um valor à propriedade) e Get (para obter o valor da propriedade).

O VB 2010 tem suporte ao Auto-Implemented Properties que cria automaticamente os métodos Get e Set da propriedade quando se usa a sintaxe a seguir:

```
Property NomedaPropriedade As Tipo
Exemplo:
Public Class Conta
        Public  Property  saldo As Decimal
End Class
Exemplo utilizando apenas atributos:
Module Module1
    Sub Main()
        Dim ponto1 As Ponto
        ponto1 = New Ponto()
        ponto1.x = 1.0
        ponto1.y = 2.0
        Console.Write("A coordenada x do ponto1 é: ")
```

```
        Console.WriteLine(ponto1.x)
        Console.Write("A coordenada y do ponto1 é: ")
        Console.WriteLine(ponto1.y)
        Console.ReadKey()
    End Sub
    Public Class Ponto
        ' definição dos membros da classe
        Public x As Double
        Public y As Double
    End Class
End Module
```

Exemplo utilizando propriedade do atributo:

```
Module Module1
        Sub Main()
        Dim conta1 As Conta
        conta1 = New Conta()
        conta1.Saldo = 1000.5
        Console.Write("O saldo da sua conta é: ")
        Console.WriteLine(conta1.Saldo)
        Console.ReadKey()
    End Sub
    Public Class Conta
        Public Property Saldo As Decimal
    End Class
  End Module
```

Métodos → São as ações (funções, procedimentos ou serviços) que os objetos da classe podem realizar.

```
Module Module1
    Sub Main()
        Dim conta1 As Conta
        conta1 = New Conta
        conta1.Saldo = 1000.5
        conta1.Exibir()
    End Sub
    Public Class Conta
        Public Property Saldo As Decimal
        Public Sub Exibir()
            Console.Write("O saldo da sua conta é: ")
            Console.WriteLine(Saldo)
            Console.ReadKey()
        End Sub
    End Class
End Module
```

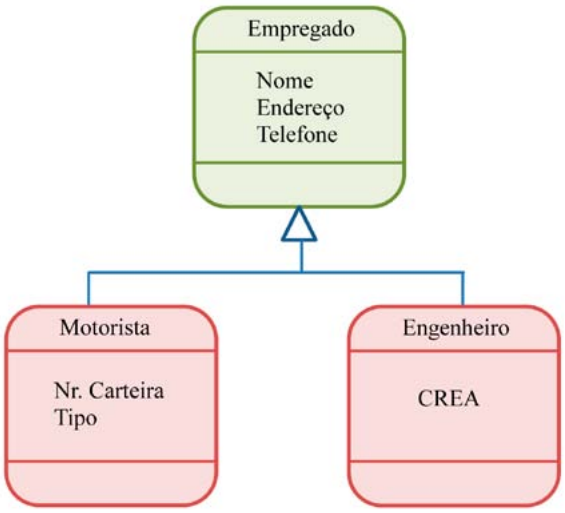
Recordando

Neste capítulo, você aprendeu a colocar em prática o conceito de programação Orientada a Objetos.

9 - ORIENTAÇÃO A OBJETOS-HERANÇA

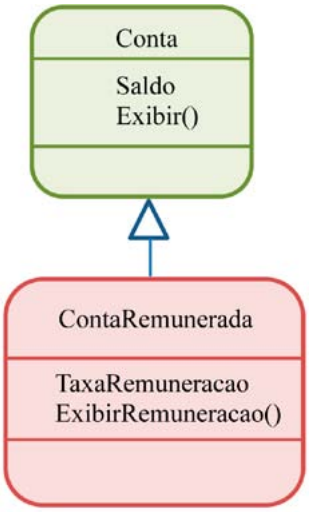
A herança permite o reaproveitamento do código entre classes utilizando-se do conceito de generalização e especialização:

- **Superclasse:** define os membros genéricos e comuns.
- **Subclasses:** herdam os membros da superclasse e definem apenas as diferenças em relação à superclasse.



Para criar uma nova subclasse, procura-se uma superclasse adequada e se codifica apenas as especializações (diferenças) na subclasse.

Esta abordagem permite aumentar a produtividade no desenvolvimento de códigos e melhorar a compreensão e o controle sobre estes, porque os agrupa em uma estrutura hierárquica.



9.1 - HERANÇA NA PRÁTICA

```
Module Module1
    Sub Main()
        Dim conta1 As Conta
```

```
        conta1 = New Conta
        conta1.Saldo = 1000

        Dim conta2 As ContaRemunerada
        conta2 = New ContaRemunerada()
        conta2.Saldo = 2000
        conta2.TaxaRemuneracao = 1.0
        conta2.Exibir()
        conta2.ExibirRemuneracao()
         Console.ReadKey()
    End Sub
Public Class Conta
        Public Property Saldo As Decimal
        Public Sub Exibir()
            Console.Write("O saldo da sua conta é: ")
            Console.WriteLine(Saldo)
        End Sub
    End Class
    Public Class ContaRemunerada
        Inherits Conta
        Public Property TaxaRemuneracao As Decimal
        Public Sub ExibirRemuneracao()
            Console.Write("A taxa de remuneração da sua conta é: ")
            Console.WriteLine(TaxaRemuneracao)
        End Sub
    End Class
 End Module
```

Recordando

Neste capítulo, você aprendeu e colocou em prática um dos conceitos mais importantes da Orientação a Objetos, a herança, que permite o reaproveitamento.

10 - BANCO DE DADOS

Segundo Silberchatz (1997) "Banco de dados é uma coleção de dados relacionados com algum significado inerente". O banco de dados então precisa de dados, que Segundo Elmasri (2005) "são fatos que possuem um significado implícito".

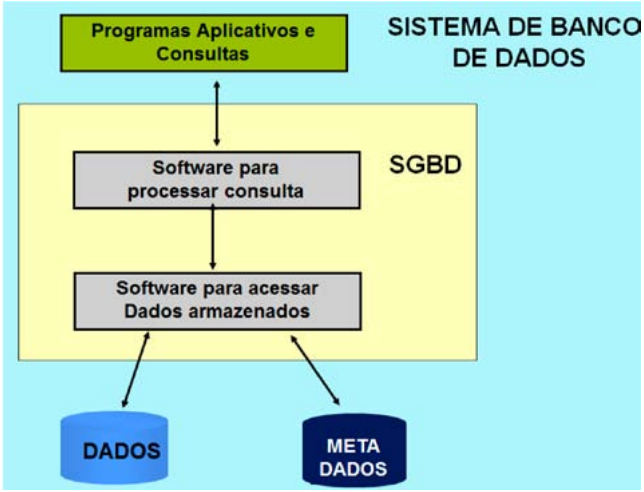
Ainda segundo Elmasri (2005) "SGBD é uma coleção de softwares de propósito geral que facilita o processo de: definir, construir e manipular um Banco de Dados (BD); situada entre uma ou mais aplicações (APs) e um ou mais BDs".

Um banco de dados possui arquitetura que permite que as aplicações compartilhem as funções especializadas existentes na coleção de programas do SGBDR e desta forma evita-se que estas funções sejam implementadas nas aplicações.

Figura 1: Banco de dados.

Segundo Silberchatz e Elmasri as principais vantagens de um SGBD são:

- Separação entre programas e dados.
- Armazenamento persistente dos dados.
- Padronização dos dados.
- Compartilhamento dos dados.
- Controle de redundância dos dados, restrições de integridade.
- Restrição de acesso.
- Eliminação de códigos redundantes nas aplicações para gerenciamento de dados, flexibilidade.
- Disponibilidade.
- Escalabilidade.



Saiba mais!

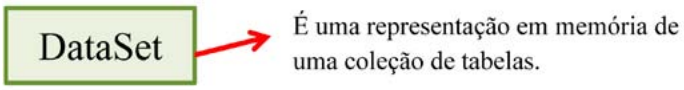
www.fgrweb.com.br
www.portaldaprogramacao.com

11 - COMPONENTES PARA BANCO DE DADOS

O SGBD permite gerenciar bancos e aceita que aplicações se acoplem a estes conjuntos de informações, mas para ter acesso, a aplicação necessita de recursos de acesso a banco de dados. O Visual Basic, disponibiliza o ADO.NET.

11.1 - BIBLIOTECA ADO.NET

Lançada em 2000 (Plataforma .NET) – Biblioteca de Acesso a Banco de Dados.
Facilitam o trabalho do programador e permitem desenvolver aplicações com maior desempenho, porque os dados são manipulados em um DataSet armazenado em memória, não necessitando acessos ao disco.



11.1.1 - VANTAGENS

A conexão com o banco de dados é realizada somente quando se deseja atualizar a fonte de dados com os dados do DataSet.

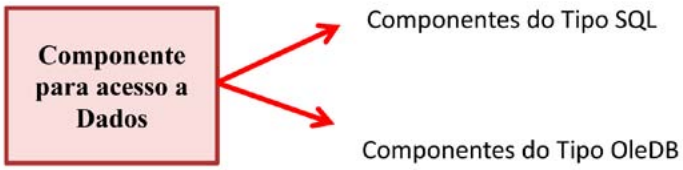
Podemos manipular os dados independentemente da localização física da fonte de dados, uma vez que a manipulação dos dados é feita no DataSet.

Podemos desenvolver aplicações em múltiplas camadas.

Quando não usa a ADO.NET, geralmente são abertas conexões com o banco de dados, que permanecem ativas durante todo o tempo em que a aplicação estiver em execução, o que consume uma grande quantidade de recursos do servidor e gera um elevado tráfego na rede de dados, prejudicando o desempenho da aplicação.

11.2 - COMPONENTES ADO.NET

Componente é um objeto com métodos pré-definidos que oferecem funcionalidades para um determinado propósito.



Componente - SQL	Componente - OleDB
Acesso, exclusivamente, aos bancos de dados do Microsoft SQL Server.	Acesso aos banco de dados que não são do Microsoft SQL Server e que tenham um provedor OleDB
Comunicam com o banco de dados de forma muito mais eficiente	Comunicam com o banco de dados de forma um pouco menos eficiente
Comunicação: Protocolo TDS (o servidor é acessado em baixo nível)	Comunicação: Camada de API (retarda um pouco a comunicação entre cliente e servidor)

11.2.1 - COMPONENTES DO TIPO SQL

SqlConnection --> Representa uma conexão com o banco de dados.
SqlCommand --> Representa um comando SQL.
SqlDataAdapter --> Representa um adaptador de dados.
SqlDataSource --> Representa uma fonte de dados.

11.2.2 - COMPONENTES DO TIPO OLEDB

OleDBConnection --> Representa uma conexão com o banco de dados.
OleDBCommand--> Representa um comando SQL.
OleDBDataAdapter --> Representa um adaptador de dados.
OleDBDataSource --> Representa uma fonte de dados.

11.2.3 - COMPONENTES PARA MANIPULAÇÃO DE DADOS

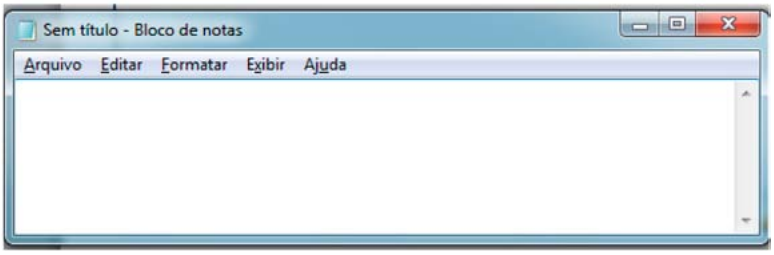
DataSet --> Representa uma coleção de tabelas em memória.
DataTable --> Representa uma tabela de um banco de dados.
DataRow --> Representa uma linha da tabela.
DataColumn --> Representa uma coluna da tabela.
DataView --> Representa uma visão de uma coleção de tabelas.

Recordando

Neste capítulo, você conheceu as ferramentas necessárias para manipular dados em um SGBD.

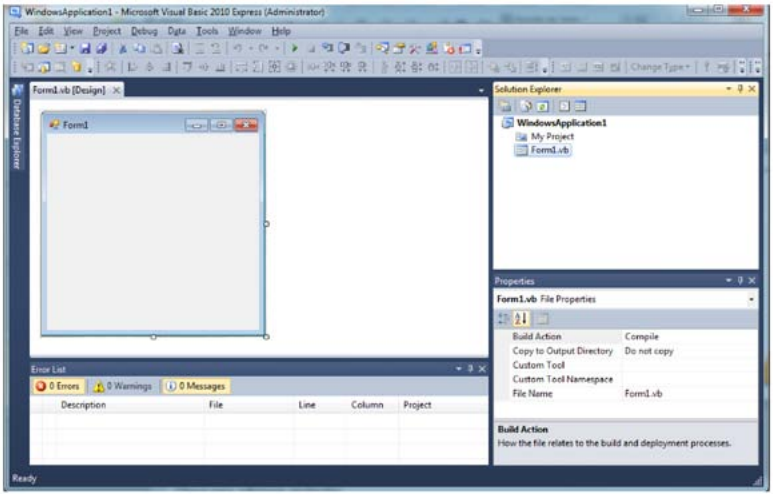
12 - GUI

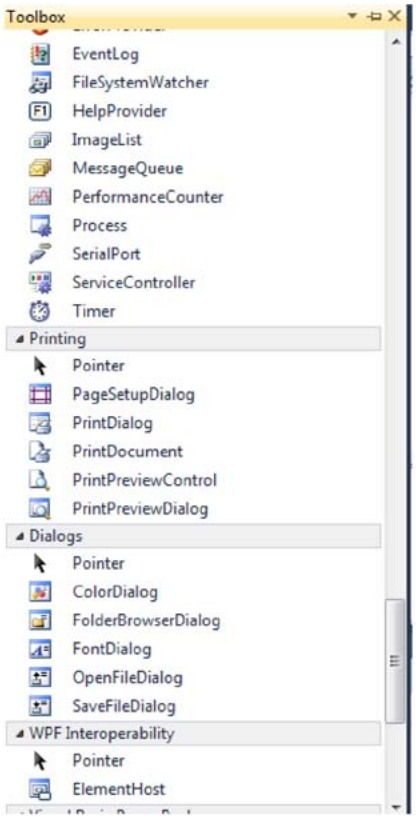
A GUI (*Grafic User Interface*) Interface Gráfica com o usuário criada na década de 80, fez com que os computadores fossem parar na mão de pessoas leigas.
Todos conhecem no Windows o sistema gráfico.



Vamos criar então nossa primeira aplicação gráfica.

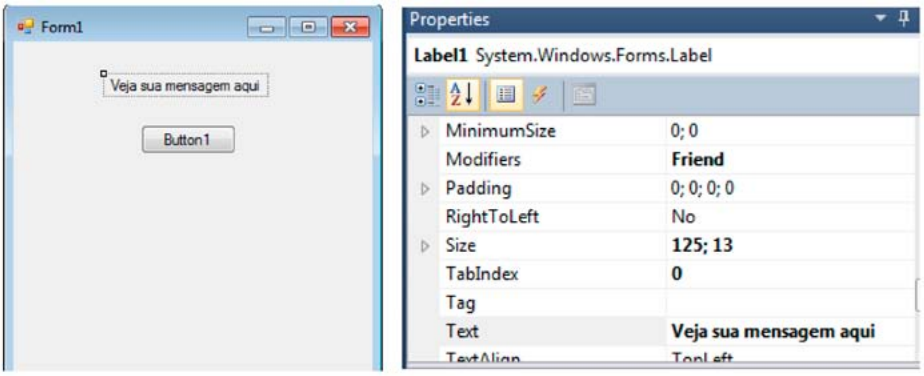
File --> New Project --> Windows Forms Applications





Ao lado esquerdo temos o Toolbox que contém os objetos necessários para criar uma aplicação gráfica, basta arrastar e soltar dentro da sua janela de aplicação.

Arraste um label para a janela e um botão.
Clique sobre a label e mude na propriedade **Text para Veja sua mensagem aqui.**



Clique duas vezes no button1.
Insira o seguinte código:
Label1.Text = "Seja bem-vindo"
Clique em executar e depois clique no botão.

Recordando

Neste capítulo, você conheceu as ferramentas gráficas necessárias para criar aplicações com interface do usuário gráfico.

Referências

Campos, Carlos Joaquim E. *Lógica de Programação Estruturada*. 1. ed. Porto Alegre: Nelpa, 2003.

Figura 1: Preferência em Linguagens de Programação. Disponível em: http://vidadehacker.wordpress.com/2011/07/11/preferencia-de-linguagens-de-programacao/. Acessado em: 21/05/2012.

Forbellone, André Luiz Villar; Eberspacher, Henri Frederico. *Lógica de Programação*. 3. ed. Person Books, 2005.

Keywords (Visual Basic). Disponível em: http://msdn.microsoft.com/en-us/library/dd409611.aspx. Acessado em: 23/05/2012.

Microsoft Visual Studio. Disponível em: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express. Acessado em: 21/05/2012.

SOUZA, Marco Antonio Furlan de; et al. *Algoritmos e Lógica de Programação*. 2. ed. São Paulo: Cengage Learning, 2005.

alfamacursos.com.br